

# Ingeniería del Software II

## Parcial #2 - Análisis Estático de Programas

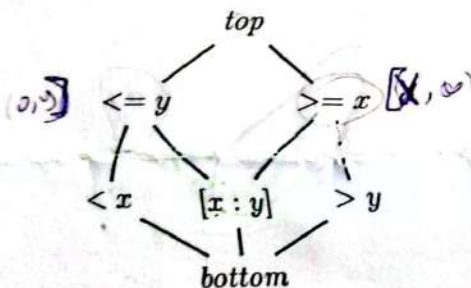
Ej1	Ej2	Ej3	Ej4	Nota
M	B	B	B	A

C: Flopy

El exámen es a libro abierto. Cada ejercicio se evaluará como Bien, Regular o Mal. Para aprobar el examen es necesario tener al menos 2 ejercicios Bien y al menos 1 ejercicio Regular.  
 Bien = 2,5p, Regular = 1,25p, Mal = 0p.

### Ejercicio 1

Sea  $R$  un reticulado y  $x, y$  números naturales tales que  $x < y$ .  $R$  es usado en el contexto de un análisis dataflow que intenta determinar si una variable que contiene números naturales es menor a cierto umbral  $x$ , mayor a cierto umbral  $y$ , o si está en el rango entre  $x$  e  $y$  durante la ejecución de un programa.



Se desea definir la operación "+" de manera que trate de modelar de la forma más precisa po-

sible la suma de dos <sup>modulos</sup> enteros para dicho reticulado. Definir la función de transición:

op1	op2	op1 + op2
$[x : y]$	bottom	
$[x : y]$	$[x : y]$	
$[x : y]$	$> y$	
$[x : y]$	$< x$	
$[x : y]$	$\leq y$	
$[x : y]$	$\geq x$	
$[x : y]$	top	
$<= y$	bottom	
$<= y$	$<= y$	
$<= y$	$[x : y]$	
$<= y$	$< x$	
$<= y$	$> y$	
$<= y$	$\geq x$	
$<= y$	top	

### Ejercicio 2

Dado el análisis del ejercicio 1 y sabiendo que es caracterizado como un análisis *Dataflow may forward*. Además, contamos con la información de que  $x = 5$  y  $y = 9$  y las siguientes funciones de transferencia para las expresiones:

▪ " $z = k$ " (con  $k$  constante):

▪ " $z = t \text{ op } k$ " (con  $k$  constante):

$$OUT[n](z) = \begin{cases} < x & \text{si } k < x \\ [x : y] & \text{si } x \leq k \leq y \\ > y & \text{si } k > y \end{cases}$$

$$OUT[n](z) = \begin{cases} bottom & \text{si} \\ & IN[n](t) = bottom \\ [x:y] & \text{si} \\ & IN[n](t) = < x \\ & \wedge k > 0 \\ > y & \text{si} \\ & IN[n](t) = [x : y] \\ & \wedge k > 0 \\ top & \text{sino} \end{cases}$$

Dado el siguiente programa:

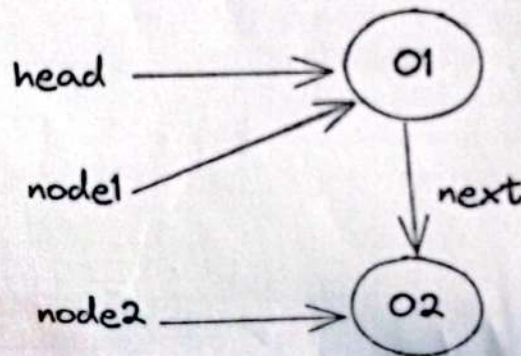
```
foo(){
  z = 4
  z = z op 5
  t = 1
  if (z >= 0) {
    t = t op 1
  } else {
    z = z op 1
  }
```

```
}
return z
}
```

- Construir el CFG del programa.
- Utilizando el análisis propuesto, calcular el valor abstracto de las variables para los conjuntos IN y OUT de cada nodo en el CFG.

### Ejercicio 3

Sea G el siguiente points-to graph para un programa P computado a partir de un algoritmo no sensitivo a flujo:



Escriba un programa P válido para G.

### Ejercicio 4

Sea el siguiente programa en Java:

```
/*@ requires x > 3;
   @ ensures: \result >= 0;
   int square_root(int x) {
     int i = 0;
     while (i * i <= x) {
       @ invariant ...
       i++;
     }
     return i - 1;
   }
```

Nota: La instrucción `return E`, asigna la expresión `E` a la variable de especificación `result`.  
Dadas  $Q := i = \sqrt{x} + 1$  la postcondición del ciclo y  $B$  la guarda del ciclo, proponer un invariante de ciclo  $I$  y probar:

- $I \wedge \neg B \rightarrow Q$
- $I \wedge B \rightarrow WP(i++, I)$



28/18

# BACON SQUAD

## Ejercicios 4.1

032	032	032 + 032
$[x:y]$	Bottom	Bottom
$[x:y]$	$[x:y]$	<del>Bottom</del> <del>top</del> $\geq x$
$[x:y]$	$> y$	$> y$ $\geq x$
$[x:y]$	$< x$	<del>top</del> $\geq x$
$[x:y]$	<del>Bottom</del>	<del>top</del> $\geq x$
$[x:y]$	$<= y$	<del>top</del> $\geq x$
$[x:y]$	$>= x$	<del>top</del> $\geq x$
$[x:y]$	top	$\geq x$
$[x:y]$	Bottom	Bottom
$<= y$	$<= y$	top $\geq x$
$<= y$	$[x:y]$	top $\geq x$
$<= y$	$< x$	top $\geq x$
$<= y$	$> y$	top $\geq x$
$<= y$	$>= x$	top $\geq x$
$<= y$	top	top

Aclaración: En mi tabla hay demoras TOP y Quiero justificar mi forma de pensar.

- En estado  $<= y$  Equivale a que puede estar en el intervalo  $[0, y]$
- El estado  $>= x$  Equivale a que puede estar en el intervalo  $[x, \infty)$

Hay un problema que tienen con intervalos  $[x, y]$  y los muy

Pocos casos en donde puede asegurar que lo suma no a estar solucio



En una y más en Ode. Por lo tanto, en todos esos casos me veo  
obligado a poner  $\pm P$  ya que puede ser cualquiera de las opciones.

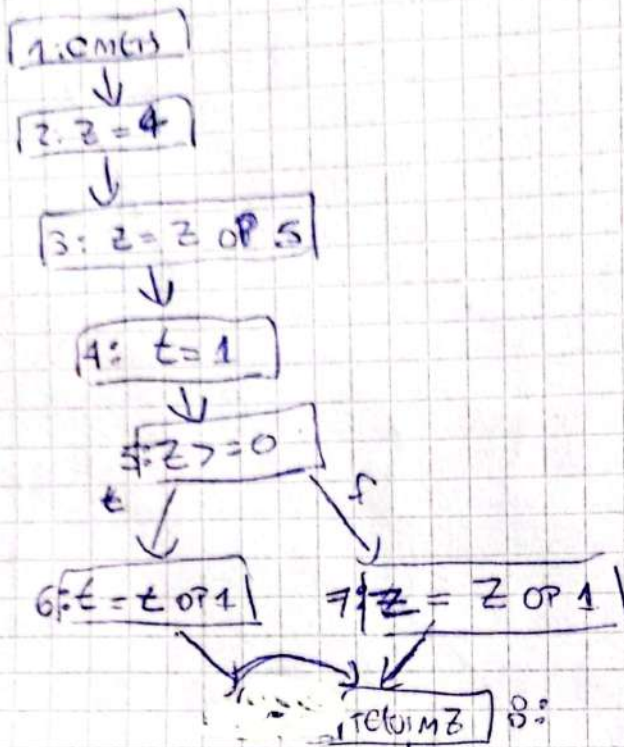


28/11/8

BOCCIO SEBASTIÁN

## Ejercicio 2:

a)



b)

mod(m)

Im(n)

Out(m)

1

-

 $\emptyset$  z: 1 t: 1

2

~~z: 1~~ z: 1 t: 1

z: &lt; x t: 1

3

z: &lt; x t: 1

z: [x: y] t: 1

4

z: [x: y] t: 1

z: [x: y], t: &lt; x

5

z: [x: y], t: &lt; x

z: [x: y], t: &lt; x

6

z: [x: y], t: &lt; x

z: [x: y], t: [x: y]

7

z: [x: y], t: &lt; x

z: [x: y], t: &lt; x

8

z: z x, t: &lt; y

z: z x, t: &lt; y

9

z: z x, t: &lt; y

-

28/7/18

### Ejercicio 3:

<sup>p</sup>  
~~max~~max() {

node1 = new O1()

node2 = new O2()

head = node1

node1.next = node2

}



# Ejercicio 4:

$$A := i = \sqrt{x} + 1$$

$$B := i * i \leq x$$

$$I := 0 \leq i \leq \sqrt{x} + 1$$

$$a) I \wedge B \Rightarrow Q$$

$$0 \leq i \leq \sqrt{x} + 1 \wedge i^2 > x \Rightarrow i = \sqrt{x} + 1 \equiv$$

$$0 \leq i \leq \sqrt{x} + 1 \wedge i > \sqrt{x} \Rightarrow i = \sqrt{x} + 1$$

$$0 \leq i \leq \sqrt{x} + 1 \wedge i > \sqrt{x} \Rightarrow i = \sqrt{x} + 1$$

no fícho  
de modo que  $0 \leq i$

Luego, sabiendo que  $i > \sqrt{x}$  y  $i \leq \sqrt{x} + 1$  la única opción posible es  $\sqrt{x} + 1$ . Luego,

$$i = \sqrt{x} + 1 \Rightarrow i = \sqrt{x} + 1$$

Esto es finalmente cierto.

$$b) I \wedge B \Rightarrow WP(i = i + 1, I) \equiv$$

$$0 \leq i \leq \sqrt{x} + 1 \wedge i^2 \leq x \Rightarrow WP(i = i + 1, 0 \leq i \leq \sqrt{x} + 1) \equiv$$

$$0 \leq i < \sqrt{x} \Rightarrow 0 \leq i + 1 \leq \sqrt{x} + 1$$

nuevamente  
Por ser  $i$  número natural

$$(0 \leq i + 1 \leq \sqrt{x} + 1) \Rightarrow 0 \leq i + 1 \leq \sqrt{x} + 1$$

Nuevamente, esto es finalmente cierto